IN THE SPECIFICATION

Please amend the specification as follows:

Please replace the first paragraph at page 1, with the following paragraph:

This application is a continuation-in-part of the following copending and commonly assigned application entitled "A SYSTEM AND METHOD FOR MANAGING TRANSACTIONS IN A MESSAGING SYSTEM", U.S. application Ser. No. 10/448,269 ~~xx/xxx,xxx~~, filed on May 29, 2003.

Please replace the paragraph beginning at page 1, line 21 with the following paragraph:

Asynchronous transfer of requests or messages between application programs running on different data processing systems within a network is well known in the art and is implemented, for example, by a number of commercially available messaging systems. These systems include IBM Corporation's MQSeries® family of messaging products, which use asynchronous messaging via queues. A sender application program issues a PutMessage command to send (put) a message to a target queue, and MQSeries queue manager programs handle the complexities of transferring the message under transactional control from the sender application to the target queue, which may be remotely located across a heterogeneous computer network. The target queue is a local input queue for another application program, which retrieves (gets) the message from this input queue by issuing a GetMessage command asynchronously from the send operation. The receiver application program then performs its processing on the message, and may generate further messages. MQSeries and IBM are trademarks of International Business Machines Corporation.

Please replace the paragraph beginning at page 12, line 7 with the following paragraph:

In FIG. 1, a data processing host apparatus 10 is connected to other data processing host apparatuses 12 and 13 via a network 11, which could be, for example, the Internet. The hosts 10, 12 and 13, in the preferred embodiment, comprise a J2EE® (Java 2 Enterprise Edition) product.

The J2EE product is running in one or more processes on each host, and is used for the transfer of messages, on channels, between message queues, defined in the same or a different process. The channels may be standard channels or fast channels. A standard channel provides assured once and once only delivery of messages. A fast channel does not provide such a delivery guarantee and as a result can provide faster transmission of messages. Host 10 has a processor 101 for controlling the operation of the host 10, a RAM volatile memory element 102, a non-volatile memory element 103 on which a log of message activity is stored, and a network connector 104 for use in interfacing the host 10 with the network 11 to enable the hosts to communicate. ~~Java is a trade mark of Sun Microsytems Inc.~~

Please replace the first paragraph at page 24 with the following paragraph:

Meanwhile, the prepare message transmitted at step 604 arrives at the remote JMS, JMS_S, and is delivered (606) to the Receiver_S. However, note that this is not necessarily the same receiver instance as the one to which the original message was delivered (at step 507 of fig. 5) but could a new instance of the same receiver. Either way, Receiver_S recognises the prepare message and does not pass it to MDB_S but instead calls prepare (607) on the local JTS, JTS_S, passing the xid of the client transaction C_1. On receipt of this request JTS_S, using the xid of the client transaction, looks up the xid of its locally created subordinate. It then stops the timer started at step 512 of figure 5 and calls prepare (not shown) on each of the participants in transaction S_1. Such participants would have registered as part of the processing of the onMessage request by MDB_S and may, for example, include one or more databases modified by the MDB S. JTS_S then consolidates the votes from each participant and sends (608) a vote message, preferably a non-persistent message, containing the vote and details of the transaction to which it applies (C_l) , back to the reply queue named in the prepare request. This message is transmitted (609) to JTS_C, preferably on a fast channel, which then delivers (610) ~~(510)~~ the message to Receiver_C.

Please replace the paragraph beginning at page 25, line 12 with the following paragraph:

Assuming that no participants voted rollback and all votes were received in a timely manner, transaction C_1 will be committed and this is shown in FIG. 7. Note that the sequence

of flows in FIG. 7 are similar to those of FIG. 6, but the prepare and vote messages of FIG. 6 are substituted for commit and committed messages, respectively, in FIG. 7. JTS_C sends (703) a commit message, preferably a persistent message, containing details of transaction C_1, to the remote queue, Q_S. The message is then transmitted (704), preferably on a fast channel, by JTS_C to the remote queue. Control then returns (705) to JTS_C, which continues by committing other participants in the transaction. Once all participants have been instructed to commit JTS_C may optionally wait for a response, such as committed, to commit messages sent out. This is only necessary if JTS_C requires to report heuristic errors (i.e.: in a two phase transaction where a participant has voted commit to prepare and subsequently failed to commit when instructed) and in which case JTS_C starts a timer which defines how long it is willing to wait for a response to commit messages. If the timer expires JTS_C can report an unknown outcome for the transaction. Meanwhile the commit message arrives at the remote system and is delivered (706) to an instance of Receiver_S. Receiver_S recognises the commit message and calls commit (707) on JTS_S, passing details of the transaction, C_1, included with the message. JTS_S obtains the id of the local transaction (S_1) and calls commit on each of its participants (not shown). Once each participant has returned, JTS_S sends ~~(706)~~ a message, preferably a non-persistent message, back to JTS_C giving its final outcome which, assuming all participants were successfully committed, is committed (708). This is transmitted (709) to the client system, preferably on a fast channel, where it is delivered (710) to Receiver_S, which passes the outcome on to JTS_C (711). Once JTS_C has obtained an outcome from each participant in the transaction it returns (712) to the client reporting the consolidated outcome.

Please replace the paragraph beginning at page 30, line 26 with the following paragraph:

However, if the target queue was found to be local at step 802, a further check is made at step 807 to see if an MDB has been registered to receive messages from the queue. If this is the case, at step 808, the message is passed directly to the MDB for processing and not added to the target queue. As a result the processing of the MDB is on the same thread and under the scope of the same transaction as the sending of the message. If step 807 found that there was not an MDB registered to receive messages from the target queue, procedure A is followed for the message

before it is added, at step 809 ~~709~~, to the queue for later retrieval. For all paths control then returns to the client application. However, note that in another embodiment, for a message sent to a local queue, if there is a receiver registered to receive messages from that queue, the client thread could be blocked whilst the message is passed directly to the receiver and the receiver processes the message. As a result control will not return to the client until the message has been fully processed and the receiver can process under the scope of the transaction under which the message was sent.

Please replace the paragraph beginning at page 36, line 14 with the following paragraph:

Figure 13 is a flow chart of a method followed by the transaction service during the processing of a message as described in figure 11. At step 1301 a request is received to begin a transaction, such as the request 15 made at step 1102 of figure 11. However, note that this step is not completed if a transaction is found to be active at step 1101 of figure 11. At step 1302 a request to register a receiver as a temporary participant in the transaction, such as the request made at step 1103 of figure **11,** is received. As part of this registration request the transaction is passed details of its superior/parent transaction. The registration request also informs the transaction not to complete until an unregister request is made for the same temporary participant and as a result at step 1303 the transaction service starts a timer and waits for the unregistration request. The timer defines how long the transaction is willing to wait for the received message to be processed. During this time the transaction service may receive requests from other non-message queue participants, such as databases which are changed as a result of the message processing. When the unregistration is received or the timer expires at step 1304 a check is made to see if it was the timer that expired. If the timer did expire the transaction is either marked rollback only or rolled back and completed at step 1305. As a result any subsequent attempts to register as a participant will be rejected and if the transaction is rolled back any participants already registered with the transaction are rolled back. If the timer was not found to have expired at step 1304, an unregister request must have been received, such as made at step 1105 of figure 11. This being the case the timer is stopped at step 1306 and the vote returned with the unregister request is checked at step 1306 ~~1308~~. If the vote was rollback, processing continues at step 1305

as discussed above. If, however, the vote was commit, at step 1308 a timer is started which defines the maximum amount of time the transaction is willing to wait for another message. If the timer expires the transaction will be rolled back. Finally at step 1309 the temporary participant is marked as unregistered and if there is a thread waiting for this event, as may the case at step 1404 of figure 14, the thread is notified.

Please replace the paragraph that begins on page 38, line 22 with the following paragraph:

Figure 15 is a flow chart of the method followed by the transaction service when processing a commit_one-phase or rollback request. At step 1501 a check is made to see if the transaction to be completed exists and is in a appropriate state to be completed. This may not be the case for example if the transaction was rolled back at step 1305 of figure 13. Assuming the transaction is in the correct state all registered participants are informed of the outcome and the transaction completes at step 1502.